(12) UK Patent Application (19) GB (11) 2 309 560 (13) A

(43) Date of A Publication 30.07.1997

(21) Application No 9602693.5

(22) Date of Filing 09.02.1996

(51) INT CL[6]
G06F 9/455

(52) UK CL (Edition O)
G4A AFS

(56) Documents Cited
EP 0458626 A2

(58) Field of Search
UK CL (Edition O) G4A AFP AFS
INT CL[6] G06F 9/455

(71) Applicant(s)
Insignia Solutions Plc

(Incorporated in the United Kingdom)

Kingsmead Business Park, London Road,
HIGH WYCOMBE, HP11 1JU, United Kingdom

(72) Inventor(s)
Wayne Plummer

(74) Agent and/or Address for Service
Mathys & Squire
100 Grays Inn Road, LONDON, WC1X 8AL,
United Kingdom

(54) Computer systems

(57) A physical computer system of a first type performs an emulation session in which it emulates a computer system of a second different type and in which an operating system is operating on the emulated computer system. Booting the emulation software and the operating system can take a long time. To reduce this problem, when the emulation session is shut-down, the values of those variables necessary to define the state of the emulated computer system at the start of the shutting-down step are stored. Then, when subsequently starting a emulation session, the values of the variables are restored from the stored values so as the restore the emulated computer system to its state at the start of the preceding shutting-down step. Typically, the storing of the variables during the shutting-down step and the restoring of the variables in the subsequent starting step can be arranged to take far less time than the time to boot-up the operating system from scratch.

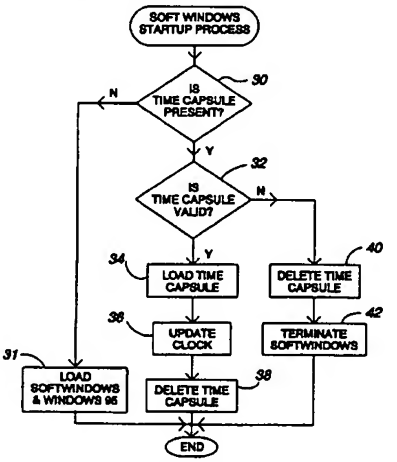

FIG.2

GB 2 309 560 A

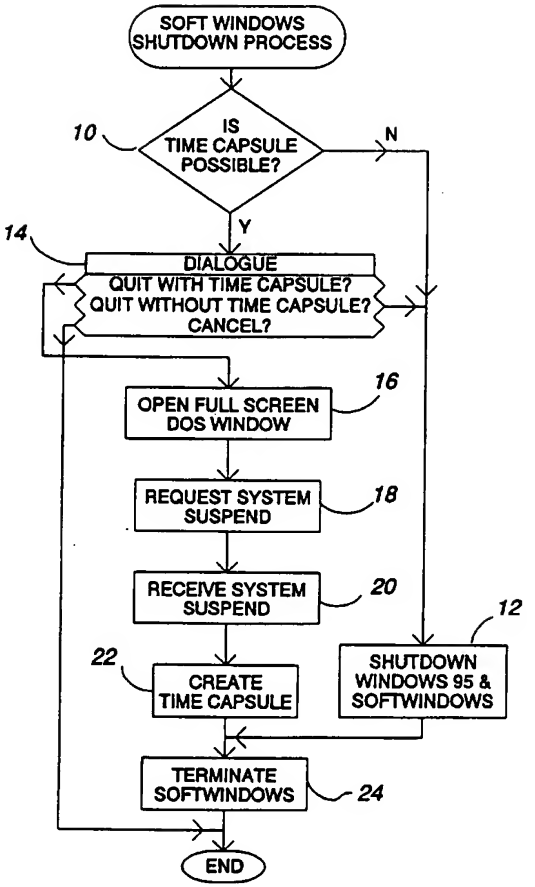
SOFT WINDOWS SHUTDOWN PROCESS
10
IS TIME CAPSULE POSSIBLE?
N
Y
14
DIALOGUE
QUIT WITH TIME CAPSULE?
QUIT WITHOUT TIME CAPSULE?
CANCEL?
16
OPEN FULL SCREEN DOS WINDOW
18
REQUEST SYSTEM SUSPEND
20
RECEIVE SYSTEM SUSPEND
12
22
CREATE TIME CAPSULE
SHUTDOWN WINDOWS 95 & SOFTWINDOWS
TERMINATE SOFTWINDOWS
24
END

FIG. 1

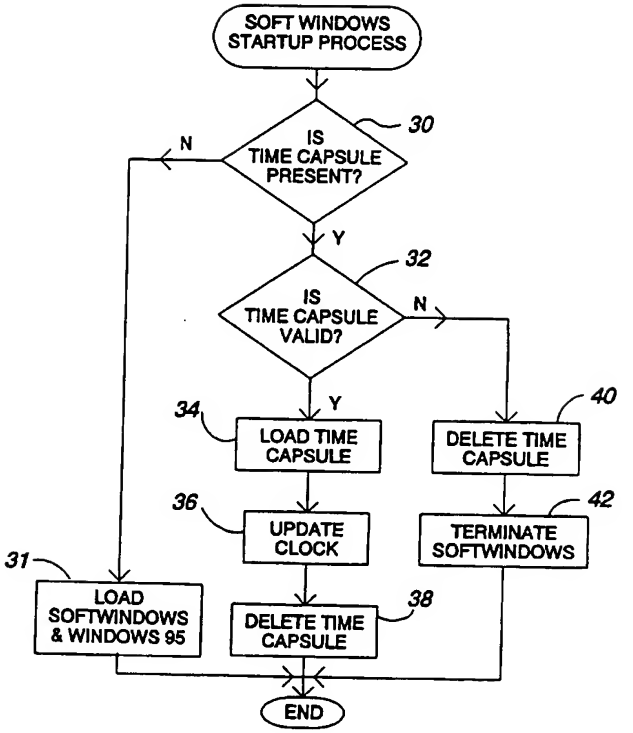
SOFT WINDOWS STARTUP PROCESS
IS TIME CAPSULE PRESENT?
30
N
Y
IS TIME CAPSULE VALID?
32
N
Y
34
LOAD TIME CAPSULE
36
UPDATE CLOCK
38
DELETE TIME CAPSULE
40
DELETE TIME CAPSULE
42
TERMINATE SOFTWINDOWS
31
LOAD SOFTWINDOWS & WINDOWS 95
END

FIG.2

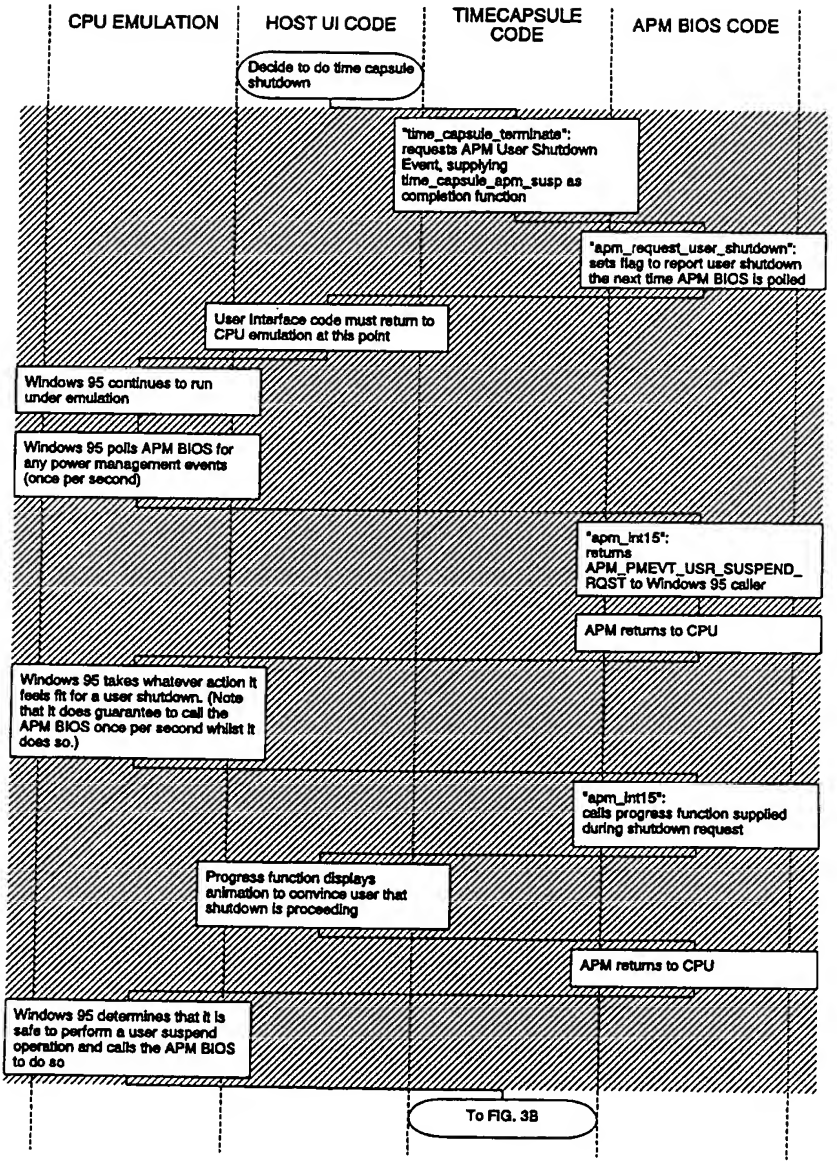
CPU EMULATION
HOST UI CODE
TIMECAPSULE CODE
APM BIOS CODE
Decide to do time capsule shutdown
"time_capsule_terminate": requests APM User Shutdown Event, supplying time_capsule_apm_susp as completion function
"apm_request_user_shutdown": sets flag to report user shutdown the next time APM BIOS is polled
User interface code must return to CPU emulation at this point
Windows 95 continues to run under emulation
Windows 95 polls APM BIOS for any power management events (once per second)
"apm_int15": returns APM_PMEVT_USR_SUSPEND_RQST to Windows 95 caller
APM returns to CPU
Windows 95 takes whatever action it feels fit for a user shutdown. (Note that it does guarantee to call the APM BIOS once per second whilst it does so.)
"apm_int15": calls progress function supplied during shutdown request
Progress function displays animation to convince user that shutdown is proceeding
APM returns to CPU
Windows 95 determines that it is safe to perform a user suspend operation and calls the APM BIOS to do so
To FIG. 3B

FIG.3A

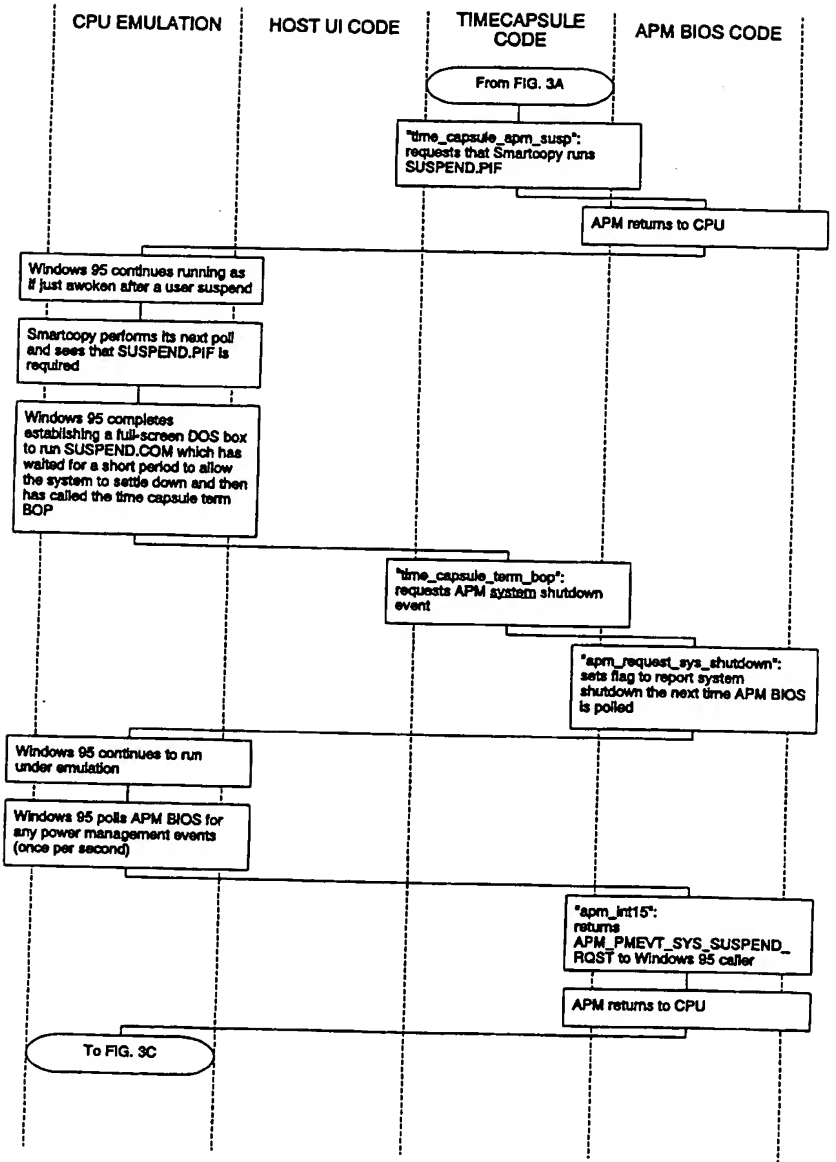
CPU EMULATION
HOST UI CODE
TIMECAPSULE CODE
APM BIOS CODE
From FIG. 3A
"time_capsule_apm_susp": requests that Smartcopy runs SUSPEND.PIF
APM returns to CPU
Windows 95 continues running as if just awoken after a user suspend
Smartcopy performs its next poll and sees that SUSPEND.PIF is required
Windows 95 completes establishing a full-screen DOS box to run SUSPEND.COM which has waited for a short period to allow the system to settle down and then has called the time capsule term BOP
"time_capsule_term_bop": requests APM system shutdown event
"apm_request_sys_shutdown": sets flag to report system shutdown the next time APM BIOS is polled
Windows 95 continues to run under emulation
Windows 95 polls APM BIOS for any power management events (once per second)
"apm_int15": returns APM_PMEVT_SYS_SUSPEND_RQST to Windows 95 caller
APM returns to CPU
To FIG. 3C

FIG.3B

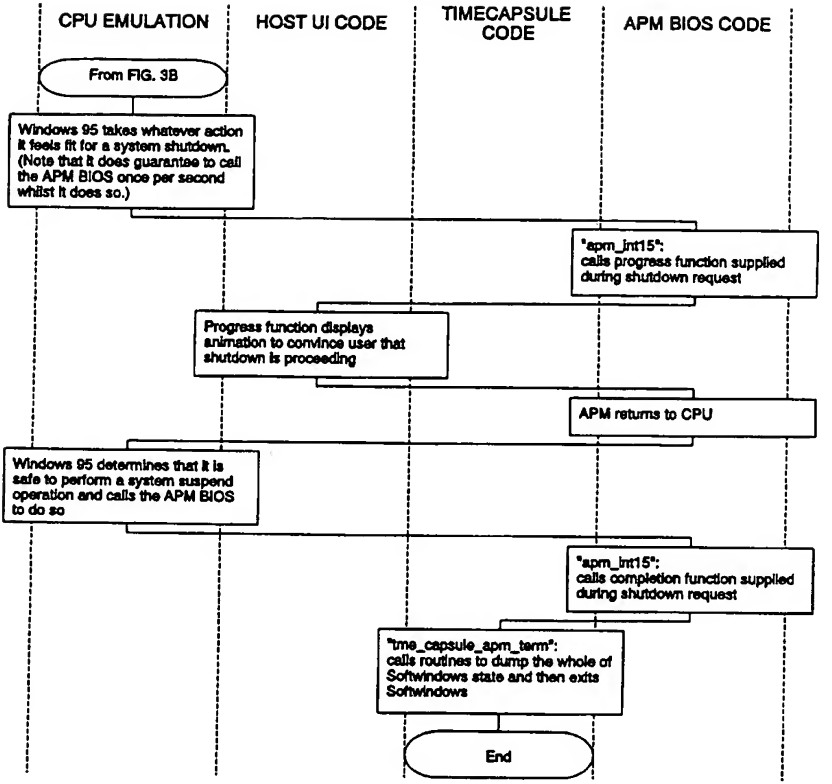
CPU EMULATION
HOST UI CODE
TIMECAPSULE CODE
APM BIOS CODE
From FIG. 3B
Windows 95 takes whatever action it feels fit for a system shutdown. (Note that it does guarantee to call the APM BIOS once per second whilst it does so.)
"apm_int15": calls progress function supplied during shutdown request
Progress function displays animation to convince user that shutdown is proceeding
APM returns to CPU
Windows 95 determines that it is safe to perform a system suspend operation and calls the APM BIOS to do so
"apm_int15": calls completion function supplied during shutdown request
"tme_capsule_apm_term": calls routines to dump the whole of Softwindows state and then exits Softwindows
End

FIG.3C

2309560

TITLE

Computer Systems

DESCRIPTION

This invention relates to computer systems and methods of operating them.

A number of different computer systems have gained great popularity over the years, examples being IBM and IBM-compatible "PCs" running the IBM-DOS or Microsoft-DOS (MS-DOS) operating system, younger generation PCs running various versions of the Microsoft Windows operating system, Apple Macintosh machines running Apple's System operating system, and various Unix machines designed primarily or exclusively to run various versions of the Unix operating system. A vast number of applications programs have been written for Windows, many of them being particularly aimed for personal use, such a personal financial programs, games, personal organisers. Also, for more general purpose programs, such as word processors, spreadsheets and the like, a greater variety of programs are commonly available for Windows than for other operating systems. This has therefore led to the problem that somebody who likes to use a particular DOS or Windows application on their home computer or notebook computer may not be able to use their homework data with the same application on their office computer unless their office computer uses the DOS or Windows operating system. Also, an application which is particularly suited to a particular job may be available to run only on the DOS or Windows operating system. If some other operating system is all that is available, it may be necessary to choose a less suitable application which can run on that operating system.

In order to deal with the problems mentioned above, emulation software has been developed, and a notable product is SoftWindows produced by Insignia Solutions of High Wycombe, GB, and Santa Clara, California, USA. One example called "SoftWindows 2.0 for Powermac" can run on an Apple Power Macintosh machine running the Apple System 7.1 operating system. This emulation software causes the physical Macintosh computer to emulate a PC having an Intel 80486DX processor and 80487 maths co-processor, 1MB of

Intel-like conventional memory and up to 2GB of Intel-like extended memory and 32MB of Intel-like expanded memory (depending upon the amount of physical Macintosh memory), two PC-like hard disks (C: and D:), a 256 colour SVGA display, an enhanced IBM-style 101-key keyboard, and a number of other features commonly provided by younger generation PCs. The emulation software package also includes MS-DOS version 6.22 and Microsoft Windows version 3.11. Accordingly, when SoftWindows is loaded on the physical Macintosh computer for a SoftWindows session, Windows applications can be run on the computer, and the computer appears to the application and to the user as if it were '486 PC.

In 1995, Microsoft launched a new version of Windows called "Windows 95." This operating system is said to provide a number of attractive features such as an improved user interface, easier customisation, easier networking, faster performance, and improved file handling and memory management. However, Windows 95 does suffer from disadvantages compared with Windows 3.11: the required and preferred memory capacities are higher; the required hard-disk space is larger; and it takes a significantly longer time to load and start up. When Windows 95 is used with emulation software, this last disadvantage is even more pronounced. In one test carried out using a 99MHz HPPA Unix platform and software emulating an Intel 40486 processor, the time taken to load the emulation software and Windows 95 was 1 3/4 minutes. When Microsoft Word from Microsoft Office 95 was then immediately started up, the total time between initiating the emulation session and being able to open a document in the word processor application was 2½ minutes.

The problem with which the present invention is concerned is to speed up the time taken between initiating an emulation session and being able to start using an application.

In accordance with one aspect of the present invention, there is provided a method of operating a physical computer system (such as, but not exclusively, an Apple Macintosh or a Unix machine) of a first type, the method comprising the steps of: performing an emulation session on the physical computer system in which the physical computer system emulates an emulated computer system (such as, but not exclusively, a PC) of a second different type and in which an operating system (such as, but not exclusively, Windows 95) is operating on the emulated computer system; shutting-down the emulation session, the

shutting down step including the step of storing the values of those variables necessary to define the state of the emulated computer system at the start of the shutting-down step; and subsequently starting a emulation session in which the values of the variables are restored from the stored values so as to restore the emulated computer system to its state at the start of the preceding shutting-down step. Typically, the storing of the variables during the shutting-down step and the restoring of the variables in the subsequent starting step takes far less time than the time to boot-up the operating system from scratch.

Preferably, the session starting step includes the steps of: determining whether or not values have been stored defining the state of the emulated computer system at the start of a preceding shutting-down step; and if not booting the operating system to an initial state. Also, the session starting step preferably includes the steps of: determining whether or not values which have been stored defining the state of the emulated computer system at the start of a preceding shutting-down step are valid; and if not: deleting those stored values; and inhibiting the starting of an emulation session using those values. One way of emulating the files (typically numbering hundreds or thousands) stored on emulated C: and D: hard disk drives of an emulated PC is to store the data in two physical "container" files on the physical computer system. If these container files are changed between shutting down an emulation session and starting a subsequent emulation session, then the variables stored during the shutting down step will not be valid for the changed container files, and problems will arise. Therefore, one example of the validity determining step is to check whether any changes have been made to the container files.

In the case where the operating system of the emulated computer system provides a graphical user interface, the shutting-down step preferably includes the step, prior to the step of storing the values of the volatile variables, of placing the emulated system in a state in which the graphical user interface is not displayed. For example, in the case of Windows 95, opening a full screen DOS box running in the foreground has the effect of forcing Windows 95 to save the state of its desktop into the emulated Intel memory, from which the values of the desktop variables can be easily stored during the shutting down step, as compared with saving the state of the display device.

In the case where the emulated computer system employs a power management system,

preferably in the shutting down step, at least one shutting-down request is made to the power management system, which in turn makes at least one shutting-down request to the operating system of the emulated computer system. Again, in the case of Windows 95 for example, performing a partial shutdown in collaboration with an advanced power management system causes Windows 95 to perform any actions it feels necessary before a long period of dormancy.

In the case where the emulated computer system employs a plurality of function pointers, preferably: a database is maintained of every function to which the function pointers can point; during the shutting down step, the values of the function pointers are encoded according to the positions of the respective functions in the database; and during the session starting step, the encoded values of the function pointers are decoded according to the functions at the respective positions in the database. This therefore deals with possible problems arising from functions being loaded at different addresses in different emulation sessions.

In accordance with another aspect of the present invention, there is provided a physical computer system of a first type, comprising: a volatile memory for storing values of variables; permanent storage means for storing values of variables; an operating system for operating on a computer system of a second different type; means for performing an emulation session on the physical computer system in which the physical computer system emulates the computer system of the second type and in which the operating system can operate on the emulated computer system; means operable to shut-down such an emulation session including storing in the permanent storage means the values of those variables in the volatile memory necessary to define the state of the emulated computer system at the start of operation of the shutting-down means; and means operable to start a subsequent emulation session including restoring from the stored values in the permanent storage means the values of the variables in the volatile memory so as to restore the emulated computer system to its state at the start of the preceding operation of the shutting-down means.

A specific embodiment of the present invention will now be described by way of example with reference to the accompanying drawings, in which:

Figure 1 is a flow diagram showing how a known SoftWindows shutdown module is modified in one embodiment of the invention;

Figure 2 is a flow diagram showing how a known SoftWindows startup module is modified in one embodiment of the invention;

Figure 3 which is made up of Figures 3A, 3B and 3C, is a more detailed flow diagram of time capsule creation and preliminary steps thereto.

In this specific description, the novel features will hereinafter be referred to by the name "time capsule." Also, in this description "base" refers to source modules which apply to all physical platforms on which the system may be run, and "host" refers to a particular physical platform and to source modules which are specific to that particular platform. "HFX" refers to Host Filesystem eXtension, which is a known feature of SoftWindows which allows directories in the host file system to appear as network drives in the emulated PC environment. "Smartcopy" refers to a known clipboard feature of SoftWindows which allows data and commands to be transferred between the host environment and the emulated environment. "APM" refers to Advanced Power Management and in particular to the features set out in the document "BIOS Interface Specification", Revision 1.1, September 1993, available from Intel Corporation, Literature Distribution Center, P.O. Box 7461, Mt. Prospect, IL 60056-7641, USA (order no. 241704-001) and from Microsoft Corporation, Hardware Vendor Relations Group, 1 Microsoft Way, Redmond, WA 98052, USA (part no. 781-110-X01).

The objective of the time capsule features in the embodiment of the invention is to side-step the problem of slow Windows 95 start-up time in SoftWindows. As mentioned above, on an otherwise unloaded 99MHz HPPA host platform, using state of the art 80486 CPU emulation technology (as of 11 December 1995), it takes one and three quarter minutes to get to the stage where it is possible to start up Windows 95 applications. Adding the time required to start up an application (e.g. a further forty seconds to start up MS Word from Office 95) yields an overall wait of two and a half minutes before the user can even consider opening the document they are planning to work on. Even if emulation performance were to double, the user would still would have an enforced wait of one whole minute

watching SoftWindows emulate code which is totally spurious with respect to their business needs.

Since there is no real hardware present in the PC being emulated by SoftWindows, all of the state of this PC is available as the content of the multitudinous variables inside the SoftWindows program itself – for example, the 16MBytes of memory in the emulated PC is, in effect, a byte array with 16,777,216 elements. Accordingly, it is, in principle, possible to allow Windows 95 to complete bootstrapping and then record the values of all of the variables necessary to remember the complete state of the emulated PC and terminate SoftWindows. At a later time, a new SoftWindows session can be started which, rather than emulating the many millions of instructions to return to the previously saved state, restores the recorded state back into the relevant SoftWindows variables and then continues the emulation process. Therefore, the Windows 95 session continues from the point it had previously reached in a fraction of the time it would take to get there from first principles.

One might imagine that this saved state could be used over and over again to get back repeatedly to the same point in the Windows 95 session. This is, in fact, only safe if other elements of the Windows 95 session's state are constant – in particular, the state of the two "container files." The container files are physical files in the physical computer system which contain the data which is used to emulate the C: and D: hard disk drives in the emulated PC computer system. It is important to realise that the saved state of the emulated PC (including Intel memory image) contains important "reflections" of the state of the C: and D: drives. There may be files open, directories cached, desktop patterns and/or software drivers loaded, and so on. The C: and D: drives are not "removable" media, so Windows 95 is not prepared for the contents to change suddenly without its knowledge. This is effectively what would be happening if an attempt were made to restore a recorded state without restoring the container files as well. In the current context, this therefore limits the usefulness of the time capsule technique to situations where the recorded state is known to be relevant to the state of the container files. In practice this means that the technique is most useful for storing a running Windows 95 session strictly from the end of one SoftWindows session to the start of the subsequent SoftWindows session.

Figures 1 and 2 illustrate an overview of how the time capsule feature is hooked into the

SoftWindows shutdown process (Figure 1) and start-up process (Figure 2).

The shutdown process is preferably invoked by the normal user interface quit mechanism rather than a separate new user interface mechanism. In the shutdown process of Figure 1, the time capsule code determines in step 10 whether a time capsule shutdown is possible or not. Time capsule shutdown is prohibited if, for example, Windows 95 not running, or if the Windows 95 session has any HFX files open for writing. If time capsule shutdown is not possible, the user will be presented with the normal SoftWindows shutdown dialogue in step 12. However, if time capsule shutdown is possible, a new dialogue is presented in step 14 which offers the user three choices: Quit with the creation of a time capsule; Quit without the creation of a time capsule; and Cancel. The two quit options display a warning message to the user. The latter one (without time capsule creation) shows the same warning as a conventional SoftWindows shutdown, i.e. that the user will lose unsaved work. The former, default quit option, displays a warning message that stresses the following points: the running Windows 95 session will be saved to disk for automatic re-start next time SoftWindows is run; the user really should save any unsaved work before proceeding (if the user decides that they want to quit Windows applications after reading the warning, they should select the Cancel option); and the restartability of the Windows 95 session at the start of the next SoftWindows session is dependent on the user not using the container files in any other way in the mean time, not altering any files that were open for reading over HFX in the meantime, and not altering their configuration in any way. If the user chooses to continue with time capsule shutdown, the time capsule creation process commences in partnership with Windows 95. This means that the SoftWindows session appears to carry on for a short while, and the user interface may be provided some visual feedback during this time. The time capsule creation system can call the user interface back once a second to drive this, for example in direct response to calls that Windows 95 makes back on SoftWindows.

In preparation for the creation of the time capsule, in step 16 SoftWindows causes a DOS program to run in the Windows 95 session; this program is set-up to run in a full-screen DOS box. This causes Windows 95 to save its desktop into the emulated Intel memory which means that SoftWindows need take no further action to save the Windows desktop, other than to save the state of the emulated Intel memory. The final call back to

SoftWindows from Windows 95 is an APM "System Suspend" call in step 20. Windows 95 makes this in response to an earlier "Request System Suspend" message being passed to it, in step 18, when it polls for any power management event. This is the ideal time for SoftWindows to create the time capsule, since Windows 95 will have performed any actions it feels is necessary before a long period of dormancy. The time capsule is then created and SoftWindows exits.

It should be noted that the APM dialogue is not allowed by Windows 95 if it detects that real mode network drivers are installed (such as used by the current version of Softnode available from Insignia Solutions). This does not prevent the time capsule shutdown from proceeding. It does proceed, but with a warning message that there is a risk of more data being lost if the time capsule becomes unusable (for example due to hard disk changes). The user then has the choice of continuing or aborting the time-capsule production, and if they abort, SoftWindows continues running Windows 95 as before.

In step 22, the time capsule is created by saving the current values of all of the volatile variables defining the state of the emulated computer system. The precise form which the time capsule takes may be dependent on the platform being used. The time capsule takes up space on the disk of the physical computer system when SoftWindows is not running. It contains the content of the whole of emulated Intel memory and other states. The Intel memory is the major part of the saved state, so the space required on disk is fairly directly controlled by the size of this. The additional states will probably never exceed 1Mbyte. Accordingly, if the size of the emulated Intel memory is 16MB, the size of the time capsule is likely to be less than 17MB. Investigations into compressing the stored version of Intel memory in the time capsule have been made, but indications are that this will significantly increase the time taken to read and write the time capsule.

Finally, in step 24, the shutdown process terminates the current SoftWindows session.

Referring to Figure 2, when SoftWindows is started up, it checks in step 30 to see whether a time capsule from a previous session is present. If not, then SoftWindows and Windows 95 are loaded from scratch in step 31 in the conventional way. However, if a time capsule is present, the startup process then checks in step 32 whether the time capsule is valid. If

so, the recorded emulated PC state from that time capsule is loaded in step 34 and the Windows 95 session resumed. In step 36, an APM "Update Time Notification" is sent to Windows 95 the next time it polls the SoftWindows APM BIOS for any power management events. This ensures that the Windows 95 time is updated from the host platform's real time clock. In step 38, the time capsule is deleted when it has been successfully loaded. It cannot have any further relevance since the user is running Windows 95 and may well be changing the contents of the C: and D: container files.

The time capsule subsystem preferably has the facility to allow the user interface code for any particular platform to determine the number of times the running Windows 95 session has been restarted from a time capsule. A value of zero means that this is a normal, first session.

If, in step 32, it is determined that the time capsule is not valid, for example because the container files for the C: and D: drives have changed, a panel explaining the problem is displayed. When the user continues from this point, the time capsule is deleted in step 40, and SoftWindows terminates immediately in step 42. If the user restarts SoftWindows again at this point, no time capsule will be found in the subsequent step 30, and Windows 95 will boot up from scratch in step 31.

There now follows a more detailed description of steps 10, 14, 16, 18, 20 and 22 of the shutdown process described above with reference to Figure 1. The time capsule shutdown is invoked by the host user interface's normal "Exit SoftWindows" mechanism. In step 10, the host user interface code uses the API with the new time capsule subsystem to determine whether it is even possible to create a time capsule at the current time, using a routine called "get_time_capsule_save_state". This routine returns two values of types "tcstat_status_value" and "tcerr_status_value" to the caller indicating the feasibility of creating a time capsule at this time. The first value, of type "tcstat_status_value", can take one of the values given in Table 1.

TABLE 1

TCSTAT_SAVE_IMPOSSIBLE	It is not possible to create a time capsule at this time, the second value gives more detail.
TCSTAT_SAVE_POSSIBLE	It is possible to create a time capsule at this time, although conditions are not ideal, the second value gives more detail.
TCSTAT_SAVE_RECOMMENDED	As far as the time capsule subsystem can tell, conditions are ideal for the creation of a time capsule.

The user interface then proceeds to call either the original SoftWindows "terminate()" function or a new "time_capsule_terminate()" function according to the values returned (possibly in conjunction with the results of further interaction with the user). The second value returned by this function, of type "tcerr_status_value", communicates details of any problem. Details of error conditions that make it impossible to create or restore a time capsule are given in Table 2.

TABLE 2

Symbolic Error Name	Description
TCERR_ APM_REQUEST_REJECTED	Windows 95 has rejected the APM request to suspend - this can happen if old-fashioned real-mode drivers are present (see note 3).
TCERR_ CMOS_CHANGED	The content of emulated CMOS RAM has been altered since the time capsule was created (see note 5).
TCERR_ CONFIG_CHANGED	The configuration of SoftWindows has been altered since the timecapsule was created (see note 5).
TCERR_ COULDNT_CLOSE	Internal Error (see note 6).
TCERR_ COULDNT_OPEN_FOR_READ	Internal Error (see note 6).
TCERR_ COULDNT_OPEN_FOR_WRITE	Internal Error (see note 6).
TCERR_ COULDNT_READ	Internal Error (see note 6).
TCERR_ COULDNT_WRITE	Internal Error (see note 6).
TCERR_ C_DRIVE_CHANGED	The container file representing the emulated C: drive has been altered since the time capsule was created (see note 5).

Symbolic Error Name	Description
TCERR_ DIFFERENT_HOST_MACHINE	Networking was active when the time capsule was created and it is being attempted to re-use the time capsule on a different machine – this is not supported because Windows 95 doesn't expect the network address to change once it's started up (see note 4).
TCERR_ DIFFERENT_SoftWindows	(see note 5).
TCERR_ D_DRIVE_CHANGED	The container file representing the emulated D: drive has been altered since the time capsule was created (see note 5).
TCERR_ ENUMERATION_FAILURE	Internal Error (see note 6).
TCERR_ HFX_CHANGED	An HFX file that was being read when the time capsule was created has been changed in the interim (see note 5).
TCERR_ HFX_FILE_IN_USE	Internal Error (see note 6).
TCERR_ HFX_SEEK_ERR	Internal Error (see note 6).
TCERR_ HFX_SYSTEM_ERR	Internal Error (see note 6).
TCERR_ MEMORY_PROBLEM	Insufficient memory (see note 4).
TCERR_ MSWDVR_ACTIVE	It is not possible to create a time capsule at this time because the Windows driver has somehow reactivated – this is normally because the user has tried to shut down SoftWindows when things were going on within the Windows 95 session – the solution is to try again when things have quietened down (see note 1).
TCERR_ NETWORK_REINIT_FAILED	Internal Error (see note 6).
TCERR_ NO_APM	The Windows 95 Session has not established an Advanced Power Management dialogue with SoftWindows' APM BIOS (see note 2).
TCERR_ NO_SMARTCOPY	The SmartCopy Windows 95 application is not running (see note 1).
TCERR_ OK	No Error
TCERR_ SCSI_CONFIG_CHANGED	(see note 4).

Symbolic Error Name	Description
TCERR_ STRING_PROBLEM	Internal Error (see note 6).
TCERR_ TEMPORARY_CPU_PROBLEM	It is not possible to create a time capsule at this time because of internal conditions within the Intel CPU emulation subsystem – this is normally because the user has tried to shut down SoftWindows when things were going on within the Windows95 session – the solution is to try again when things have quietened down (see note 1).
TCERR_ WRITABLE_HFX	An application is currently in the process of writing to an HFX drive (see note 1).
TCERR_ WRONG_SMARTCOPY	An earlier version of SmartCopy is running than one that will properly support time capsule shutdown (see note 1).

The significance of the notes in Table 2 is given in Table 3.

TABLE 3

1	This error can cause get_time_capsule_save_state to return TCSTAT_SAVE_ IMPOSSIBLE.
2	This error can cause get_time_capsule_save_state to return TCSTAT_SAVE_ POSSIBLE (as opposed to the preferred TCSTAT_SAVE_RECOMMENDED).
3	Although this error doesn't affect the result of get_time_capsule_save_state(), it can occur later in a time capsule shut-down; this error doesn't prevent the time capsule from being created, but it will be a more risky time capsule given that APM could not be used.
4	This error can occur when loading a time capsule and prevents successful loading of the time capsule.
5	This error can occur when loading a time capsule and prevents successful loading of the time capsule (i.e. as above) – but in-house within Insignia Solutions Plc., this error can be configured to be ignored for testing purposes.
6	Unexpected error.

Steps 16, 18, 20 and 22 shown in Figure 1 are initiated by a routine called "time_capsule_terminate." This routine is the one that the host user interface code calls to actually initiate the shutting-down of SoftWindows with the creation of a time capsule. The single input parameter is a pointer to a routine the host user interface code wishes to have called once a second to provide visible feedback to the user that shutdown is proceeding. The process of shutting down SoftWindows and creating a time capsule is relatively

complex (in comparison with a traditional shutdown) and involves interaction with Windows 95 and some Windows 95 applications. It is should be noted that this call just kicks off the whole process – it does return (and relatively quickly, at that). The time capsule shutdown process depends on the user interface code returning to the CPU emulation after having called "time_capsule_terminate." Other routines called "time_capsule_apm_susp", "time_capsule_term_bop" and "time_capsule_apm_term", all co–operate in the process of getting the Windows 95 session into an appropriate state for saving into the time capsule as shown in Figure 3, which is hereby incorporated into the description. One significant requirement of the state Windows 95 should be in (in order to facilitate the saving into a time capsule) is that the Windows desktop be completely obscured by a full–screen DOS box running in the foreground. This has the desirable effect of forcing Windows 95 itself to save the state of the Windows 95 desktop into Intel memory (which is trivial to save into the time capsule in comparison to the detailed saving of the display device's state).

In an early prototype of the invention, the time taken to shutdown SoftWindows was sometimes excessive. The main culprit turning out to be SUSPEND.COM itself which was busy waiting in the full–screen DOS box and not allowing Windows 95 to proceed. In order to alleviate this problem, SUSPEND.COM may be arranged to perform VM_RELEASE_TIMESLICE calls. In order to improve further the speed of operation and if no software is found which takes specific advantage of the User Shutdown APM dialogue, then all of the steps shown on the shaded background in Figure 3 may be omitted.

The final routine shown in Figure 3, "time_capsule_apm_term", calls a routine called "save_SoftWindows_state" to save the state of SoftWindows. It uses the same host time capsule manipulation routines as used to open the time capsule (and as described below) for writing and then calls the save routines in each SoftWindows subsystem in turn. Each subsystem's save routine saves the necessary global state in order to be able to continue operating successfully at the start of the subsequent SoftWindows session with the software still running in the Windows 95 session. None of the state saved must rely on anything that cannot be guaranteed to be still valid at the start of the next session (e.g. the load address of the SoftWindows executable). Since function pointers are frequently used in various SoftWindows subsystems, the time capsule subsystem provides a mechanism to allow the conversion of such pointers into safe ordinal numbers and back again, as described below.

The save routines for the SoftWindows subsystems preferably do not damage the subsystems they are saving. It is quite possible, for example, that the creation of the time capsule might fail through insufficient disk space. In such a case, SoftWindows should preferably be able to continue running the user's Windows 95 session. If an error condition occurs at any stage during the creation of the time capsule, a time capsule routine called "flag_time_capsule_error" is called with the appropriate error code. This does not actually divert the flow of control during time capsule creation. A check is made at the end of time capsule creation to see if any error was flagged. If so, the first such error is reported to the user *via* a call on a host routine called "host_time_capsule_write_status" which informs the user of the problem. After the user acknowledges the existence of the problem, SoftWindows continues to execute as it did before the shutdown attempt was made. The user can then attempt to resolve the problem and try again or alternatively can quit Windows 95 and shutdown without creating a time capsule.

There now follows a more detailed description of the loading of a time capsule as shown at steps 30 to 34 in Figure 2. When a SoftWindows session starts up, much of the usual initialisation code is performed regardless of the existence of a saved session in a time capsule. To be precise, all code up to just before the end of the "reset()" function that gets called by "Reset BOP" in the SoftWindows emulated ROM gets executed. At this point, the majority of emulated hardware components have been initialised and the video subsystem is in its standard mode 3 configuration (80x25 character cells), the same as was in force when the time capsule was created (because of the full-screen DOS box that was running in the foreground, also in mode 3). At this time, a time capsule subsystem routine called "time_capsule_initialise()" is called from within "reset()." If no time capsule is found, this routine returns without doing anything significant and SoftWindows starts up as normal. If a time capsule is found, then a routine called "load_SoftWindows_state()" is called to load the saved SoftWindows state from the time capsule.

The "load_SoftWindows_state()" routine is local to the time capsule subsystem. It performs the converse operation to that of "save_SoftWindows_state()" described earlier, that is, it calls the time capsule load routines of each SoftWindows subsystem in turn, in the same order that the save routines were called during time capsule creation. Function pointers loaded from the time capsule need converting back into live function pointers, as described

in further detail below. Some subsystems will need to allocate memory to accommodate the information that is coming out of the time capsule; the success of all such allocation requests is checked, and if any fail, the routine "flag_time_capsule_error()", mentioned above, is called with the error code "TCERR_MEMORY_PROBLEM" (see Table 2).

There now follows a description of routines provided by the time capsule subsystem for use in the save/load routines for individual SoftWindows subsystems, other than the routines for handling the conversion of function pointers to/from time capsule safe values, which are described later. The routine "flag_time_capsule_error" mentioned above flags that an error has occurred. It can be called during time capsule creation or time capsule loading, but the behaviour is different in the two environments. During reading, a call on this routine reports the error to the user *via* "host_time_capsule_read_status ()". The value for "must_quit" that gets passed to this routine is determined from a table in a header file "timecap.h" that indicates which error conditions are fatal during time capsule loading. During writing, the error value is noted if it is the first error to occur, and subsequent ones are ignored. The value noted can be interrogated by calling a routine "timecapsule_error_status()". The routine "timecapsule_error_status" returns any error status that has been flagged during time capsule creation. The routine "doing_time_capsule_shutdown" returns a boolean value indicating whether a time capsule shutdown is currently in progress. The routine "get_time_capsule_vintage" returns a value that indicates the number of times that the current session has been restored from a time capsule. A value of zero indicates that no time capsule loads have been performed. It is useful that this information be available *via* the user interface somewhere to help technical support personnel determine whether problems encountered in the field can have any time capsule component eliminated from enquiries. (If it happens at vintage 0, it is almost certainly not a time capsule error.)

There now follows a description of the mechanism provided by the time capsule subsystem to facilitate the storing and reloading of function pointers to/from the time capsule. This functionality relies on a database of all functions that can ever be pointed to by variables that need to be saved/restored to/from the time capsule. The database is actually distributed amongst all relevant SoftWindows source files. The entry for any particular function is in a table in the same source module in order to avoid having to make functions global that were not before time capsule functionality was implemented. The format of the time capsule

function database for an imaginary SoftWindows source module XXX is as follows, the name "fnar" denoting function array):

```
#ifdef TIME_CAPSULE
#include "timecap.h"
/*
 * Export details of all functions in this module that can ever get pointed to
 * by variables that get put into the time capsule:
 */
timecapFuncNode XXX_fnar[] =
                {
                        tcapFuncNode (local_func_1),
                                :       :
                        tcapFuncNode (local_func_N),
                        tcapFuncNodeTerminator
                };
#endif /* TIME_CAPSULE */
```

This table is hooked into the time capsule database by the inclusion of the appropriate declaration in the appropriate header file and the appropriate reference in the appropriate "fnarar" (Array of Function Arrays) as follows:

For a Base file:

timecap.h and base_fnarar in $MASTER/base/support/timecap.c

For a Host file:

host_tcap.h and host_fnarar in the appropriate host directory.

Having included all functions in the time capsule function database by including such tables in all relevant SoftWindows modules, it is now possible to use two subroutines "tcap_encode_fn_ptr" and "tcap_decode_fn_ptr" to convert function pointers to and from numbers which are safe to include in a time capsule. The routine "tcap_encode_fn_ptr" is used to convert a function pointer *in situ* to a number that is safe to include in a time capsule. A function pointer is not safe, because when a time capsule is reloaded in a subsequent session, the function may be loaded at a different host address. The single parameter is the address of the function pointer variable to convert. This routine works simply by traversing the time capsule function database, counting as it goes and stopping when it finds the correct entry. The count value that is reached when the right entry is found is written over the function pointer whose address was supplied to "tcap_encode_fn_ptr". Note that a null pointer always returns a count of zero. Preferably, the values returned are all biased by a large 32-bit number to make then more easily recognisable in a time capsule for debugging purposes and to reduce the possibility of a small positive integer being confused for a converted function pointer in error situations. Note that if a subsystem

converts one of its function pointer variables *in situ* for writing to a time capsule during a time capsule shutdown, it must immediately replace it by calling "tcap_decode_fn_ptr", below, or by restoring a saved value if it prefers. This is in case the time capsule creation process fails and SoftWindows needs to continue running. It is a fatal error if the value in the function pointer variable is not found in the database.

The routine "tcap_decode_fn_ptr" performs the converse operation to "tcap_encode_fn_ptr", above. It traverses the time capsule function database in the same manner, again counting as it goes. When its count reaches the value that is in the function pointer variable whose address was passed, taking into account the bias that had been applied to it, the function pointer from the database entry reached is written to the function pointer variable whose address was passed.

There now follows a description of the support each host must provide to allow the base time capsule features to work. The base only accesses the time capsule through these routines and makes no assumptions on the precise nature of the time capsule other than those stated here.

Firstly, the routine called "host_open_write_time_capsule" requests that the host opens a new time capsule for writing, overwriting any existing time capsule and creating a new one if required. The time capsule is written sequentially, so that no random access is required.

The routine "host_write_string_time_capsule" writes the NULL terminated string supplied to the time capsule.

The routine "host_write_data_time_capsule" writes the requested number of bytes to the time capsule from the specified address. Note that if any error is reported *via* "flag_time_capsule_error()", by the base or the host, during the creation of a time capsule, the flow of control is not diverted; each subsystem is still called to dump its data to the time capsule. This means that the host write routines must be able to tolerate (i.e. ignore) calls made from the base even after a host error condition (such as disk full) has occurred. Any tidying up after the error condition may be left until the call on "host_close_time_capsule()" if desired.

The routine "host_write_var_time_capsule" does not need to be provided by the host, it is actually a macro in "timecap.h", but is included here for completeness.

The routine "host_time_capsule_write_status" talks to the host user interface code to make an appropriate error message get displayed on the screen to explain to the user what went wrong with time capsule creation.

The routine "host_open_read_time_capsule" attempts to open an existing time capsule for reading. A value of TRUE should be returned if the time capsule exists. Note that this is TRUE even if it is unreadable in some way. If this is the case, the host should note that there is a problem by calling the base routine "flag_time_capsule_error()" with the error code "TCERR_COULDNT_OPEN_FOR_READ" (see Table 2) and return TRUE all the same. The routine should only return FALSE if there is no time capsule found and a normal SoftWindows boot is required.

The routine "host_check_string_time_capsule_err" checks the length of the supplied string and reads exactly that number of bytes from the time capsule; it then compares the bytes read with the string. If the two differ, "flag_time_capsule_error()" is called with the error value supplied. If any other errors reading from the time capsule, "flag_time_capsule_error()" is called with the appropriate error code from Table 2.

The routine "host_check_string_time_capsule" does not need to be exported from the host; it is another macro in "timecap.h." This is intended for use when checking for strings in the time capsule that act as "anchors" or known points in the time capsule. The failure to match would then indicate that the loading of the time capsule has got "out of phase" with the data stored within it.

The routine "host_read_data_time_capsule" reads the requested number of bytes from the time capsule to the specified address. If any other errors reading from the time capsule, "flag_time_capsule_error()" is called with the appropriate error code from the Table 2.

The routine "host_read_var_time_capsule" does not need to be provided by the host. It is actually a macro in "timecap.h", but is included here for completeness.

The routine "host_time_capsule_read_status" talks to the host user interface code to make an appropriate error message get displayed on the screen to explain to the user what went wrong with time capsule loading. If the parameter "must_quit" is true, then this routine must terminate the SoftWindows run immediately the user has acknowledged the error. It is also called when the time capsule is loaded OK. The code supplied in this case is "TCERR_OK", and "must_quit" is false.

The routine "host_close_time_capsule" closes the time capsule. This routine is used whether the time capsule was being written or read.

The routine "host_delete_time_capsule" requests that the host delete the time capsule.

The routine "host_save_to_time_capsule" is called during time capsule creation and allows each host to call the time capsule save routines for any host modules that need to save state.

The routine "host_load_from_time_capsule" is called during time capsule loading and allows each host to call the time capsule load routines for any host modules that need to load state. Obviously the calls in this routine should be in a corresponding order to those in "host_save_to_time_capsule()".

Having described in detail one embodiment of the invention, it should be noted that many modifications and developments may be made thereto. Accordingly, the embodiment described above with reference to the drawings should not be taken as limiting the scope of the invention. For example, the invention may be employed on other physical computer systems emulating computer systems other than PCs and running operating systems other than Windows 95, for example other versions of Microsoft Windows, Solaris, Nextstep, Linux, and so on.

CLAIMS

1. A method of operating a physical computer system of a first type, the method comprising the steps of:

performing an emulation session on the physical computer system in which the physical computer system emulates an emulated computer system of a second different type and in which an operating system is operating on the emulated computer system;

shutting-down the emulation session, the shutting down step including the step of storing the values of those volatile variables necessary to define the state of the emulated computer system at the start of the shutting-down step; and

subsequently starting a emulation session in which the values of the volatile variables are restored from the stored values so as to restore the emulated computer system to its state at the start of the preceding shutting-down step.

2. A method as claimed in claim 1, wherein the session starting step includes the steps of:

determining whether or not values have been stored defining the state of the emulated computer system at the start of a preceding shutting-down step; and

if not booting the operating system to an initial state.

3. A method as claimed in claim 1 or 2, wherein the session starting step includes the steps of:

determining whether or not values which have been stored defining the state of the emulated computer system at the start of a preceding shutting-down step are valid; and

if not:

deleting those stored values; and

inhibiting the starting of an emulation session using those values.

4. A method as claimed in any preceding claim, wherein:

the operating system of the emulated computer system provides a graphical user

interface; and

the shutting-down step includes the step, prior to the step of storing the values of the volatile variables, of placing the emulated system in a state in which the graphical user interface is not displayed.

5. A method as claimed in any preceding claim, wherein:

the emulated computer system employs a power management system; and

in the shutting down step, at least one suspension request is made to the power management system, which in turn makes at least one suspension request to the operating system of the emulated computer system.

6. A method as claimed in any preceding claim, wherein:

the emulated computer system employs a plurality of function pointers;

a database is maintained of every function to which the function pointers can point;

during the shutting down step, the values of the function pointers are encoded according to the positions of the respective functions in the database; and

during the session starting step, the encoded values of the function pointers are decoded according to the functions at the respective positions in the database.

7. A method as claimed in any preceding claim, wherein the physical computer system is other than a PC, the emulated computer system is a PC, and the operating system is Windows.

8. A method of operating a physical computer system, substantially as described with reference to the drawings.

9. A computer system adapted to perform the method as claimed in any preceding claim.

10. A physical computer system of a first type, comprising:

a volatile memory for storing values of variables;

a permanent storage means for storing values of variables;

an operating system for operating on a computer system of a second different type;

means for performing an emulation session on the physical computer system in which the physical computer system emulates the computer system of the second type and in which the operating system can operate on the emulated computer system;

means operable to shut-down such an emulation session including storing in the permanent storage means the values of those variables in the volatile memory necessary to define the state of the emulated computer system at the start of operation of the shutting-down means; and

means operable to start a subsequent emulation session including restoring from the stored values in the permanent storage means the values of the variables in the volatile memory so as to restore the emulated computer system to its state at the start of the preceding operation of the shutting-down means.

11. A system as claimed in claim 10, wherein the session starting means includes:

means for determining whether or not values have been stored in the permanent storage means defining the state of the emulated computer system at the start of the preceding operation of the shutting-down means; and if not for booting the operating system to an initial state.

12. A system as claimed in claim 10 or 11, wherein the session starting means includes:

means for determining whether or not values which have been stored defining the state of the emulated computer system at the start of the preceding operation of the shutting-down means are valid; and if not: for deleting those stored values; and for inhibiting the starting of an emulation session using those values.

13. A system as claimed in any of claims 10 to 12, wherein:

the operating system of the emulated computer system provides a graphical user

interface; and

the shutting-down means includes means operable, prior to storing of the values of the volatile variables, to place the emulated system in a state in which the graphical user interface is not displayed.

14. A system as claimed in any of claims 10 to 13, wherein:

the emulated computer system employs a power management system; and

the shutting down means includes means to make at least one suspension request to the power management system, and the power management system includes means, in turn, to make at least one suspension request to the operating system of the emulated computer system.

15. A system as claimed in any of claims 10 to 14, wherein:

the emulated computer system employs a plurality of function pointers;

a database is maintained of every function to which the function pointers can point;

the shutting down means includes means to encode the values of the function pointers according to the positions of the respective functions in the database; and

the session starting means includes means to decode the encoded values of the function pointers according to the functions at the respective positions in the database.

16. A system as claimed in any of claims 10 to 15, wherein the physical computer system is other than a PC, the emulated computer system is a PC, and the operating system is Windows.

17. A physical computer system, substantially as described with reference to the drawings.

The Patent Office

24

Application No:	GB 9602693.5	**Examiner:**	Paul Nicholls
Claims searched:	1 - 17	**Date of search:**	12 April 1996

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.O): G4A (AFP, AFS)

Int Cl (Ed.6): G06F 9/455

Other:

Documents considered to be relevant:

Category	Identity of document and relevant passage	Relevant to claims
A	EP 0,458,626 A2 (FUJITSU) - Whole document	1, 10

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.